Designing Anaylsis and Synthesis Filterbanks in GNU Radio

Thomas W. Rondeau
University of Pennsylvania & Rondeau Research
Shelburne, VT
tom@trondeau.com

Timothy J. O'Shea
The Hume Center
Virginia Tech
Arlington, VA
oshea@vt.edu

***Index Terms*—GNU Radio, software radio, SDR, dynamic spectrum access, DSA, filters, polyphase filterbanks, analysis filterbanks, synthesis filterbanks, dsp**

***Abstract*—**

This paper provides a look at the polyphase analysis and synthesis filterbanks and filter design tools in GNU Radio. The filterbanks are two powerful blocks for performing signal detection, collection, and analysis operations that are of increasing interest to the cognitive radio (CR) and dynamic spectrum access (DSA) communities. We look at how to design and use the filterbanks to channelize, synthesize, and reconstruct signals. We use a few simple examples for explanatory purposes but which are readily available to experiment with and learn from. The processes, descriptions, and tools used in this paper are designed to allow others to adapt filtering and filterbank tools of GNU Radio for other, more complex purposes and applications.

I. Introduction

Cognitive radio (CR) and dynamic spectrum access (DSA) problems generally involve discovery and then use or reuse of spectrum. After the identification of available spectrum, the DSA system will seek to transmit and receive in it. These spectrum opportunities, however, may be close to other users, and so we risk receiving or creating adjacent channel interference with other users. Even in the discovery process, we desire efficient structures for isolating and testing areas of spectrum. Polyphase analysis and synthesis filterbanks are useful tools for these problems. And while there is a nice body of literature about how the filterbanks themselves work, the design of the filter taps is still not as widely understood.

GNU Radio as a software radio platform provides a number of tools to build, use, and analyze polyphase filterbanks for use in DSA systems. We have the principle tools necessary to construct analysis and synthesis channelizers as well as a large suite of filter design tools. Use of these, however, is not necessarily straight-forward.

In this paper, we discuss the principles of using the filterbank blocks, the FIR filter design tools, and how to build the necessary prototype filters required to effectively use polyphase filterbanks in GNU Radio. Design decisions for each step are explored to enable further use of these tools and techniques.

As an extension to the filter design process, we will also look into the process of building and using perfect reconstruction filters in GNU Radio. A reconstruction filter allows us to channelize spectrum with an analysis filterbank and then recombine any set of channels with a synthesis filterbank with no loss of signals split between channels. By combining the analysis and synthesis filterbanks like this, we can create arbitrary bandwidth channelizers.

A companion webpage[1] is available to download the flowgraphs and filters discussed in this paper.

II. Assumptions

As we cover the topics of polyphase analysis and synthesis filterbanks in GNU Radio, we recognize that they are fairly advanced topics in signal processing. Using these tools requires quite a bit of knowledge to appropriately set the parameters for an application. Here, we will give a very basic understanding of the theory and properties of the filterbank tools, but we will mostly leave the real theoretical details to other papers referenced through the text that discuss these concepts in-depth. Instead, we want to provide context about the use of these filterbanks in GNU Radio to help enable their use elsewhere. As such, we assume a basic literacy of the GNU Radio[2], such as the what it is, what it does, and the basic principles of

[1] http://www.trondeau.com/examples/2014/1/23/pfb-channelizers-and-synthesizers.html
[2] http://gnuradio.org

a *flowgraph* and a GNU Radio *block*. We also assume a basic familiarity here with digital filtering [1].

When discussing specific GNU Radio blocks, we mention the full C++ namespace of the block to make it easy to find the entry in the GNU Radio Manual[3].

We further assume that the radio front-end has sufficient selectivity and dynamic range in its out-of-band filtering and digitization to allow the desired signal to be negligibly distorted by hardware. Given this condition, we can do the rest of our signal analysis, recovery, and isolation using software filters.

With the highly dynamic nature of software filters, we expect these rapidly reconfigurable and efficient channelizer techniques to be widely applicable and an appealing option in the development of next generation cognitive and dynamic spectrum radio systems.

III. Polyphase Filterbanks: Analysis and Synthesis Filters

We begin with the basic structures of the analysis and synthesis filterbanks. The one is basically the inverse of the other. Both involve first the design and then the partition of a *prototype filter*, where the partitioning occurs over a bank of smaller filters, hence the name *filterbank*. There is then the filtering of the signal through the bank of filters and combining the filter output such that they separate or combine the signals into the different channels, practically done using a discrete Fourier transform. Details of the filterbank structures can be found in [2]. Here, we focus on the development and understanding of the prototype filter.

One of the fundamental concepts required when understanding what is going on in the channelizer filterbank is the ability to analyze the filters in terms of phase. In Figure 1, we have created a prototype filter and partitioned it among four filters ($M = 4$). The partitioned filters are simply taking every $M = 4$ tap starting at tap m where m is the channel number. We can think of this as each of the channels is the same filter with a phase delay of $m\frac{2\pi}{M}$.

A consequence of the partitioning process is that we are down-sampling the filter. Each partitioned filter is now at $1/M$ the rate of the original filter. This clues us in to both the nature of the operations being performed as well as the design methods for making the prototype filter. In the application of the filterbank, we successively insert samples into the filters, which means that each filter is processing a down-sampled version of the signal. Because of the down-sampling of the signal, we design the prototype filter at the full signal sample rate. In a channelizing filterbank, the sample rate for the design process is the input rate, and for a synthesis filterbank, we work with the output sample rate. This value turns out to be the highest sample rate the system will ever operate at.

Analysis, or channelizer, filterbanks bring in a wide spectrum and split it into equally-space, equal bandwidth channels. Think of splitting the US FM broadcast band from 88 to 108 MHz into the one hundred 200 kHz FM stations using a single filterbank. Each station is 200 kHz wide and separated by 200 kHz for each channel. Instead of filtering and moving from its RF channel to baseband for each channel, we process all one hundred channels at the same time in the channelizer.

The channelization process uses the concept of aliasing. While we generally work hard to avoid aliasing, in the channelizing filterbank, we are aliasing in a very controlled way such that we can pull out the individual channels. We know from sampling theory that when we down-sample a signal by a factor of M we also create M *Nyquist zones* that are aliased onto baseband. We typically avoid the effects of aliasing by removing energy from the parts of the spectrum that will alias by using a low pass or band pass filter. If we filter out enough of the signal, the aliased zones will not fold in and distort our desired signal.

In the channelization process, we want those aliases. Recall that the filterbanks use the same filter with a different phase. When filtering, each of the aliased zones is processed by each arm of the filterbank, which has its own phase. At this point, the output of each filterbank's arms is a set of aliases from the different channels. We can then *despin* by applying a correction factor for filter m of $exp(j2\pi km/M)$ for a given aliased zone k. The correction factor is related to the phase difference between each of M filter arms. If we despin with this function for a given k over all arms of the filterbank and sum them together, we constructively add the alias from that channel while destructively canceling all other aliases from the other channels. If we think about that correction factor and summing all arms together over every arm and every channel, we make the leap to see that this is actually a discrete Fourier transform (DFT) operation that we can efficiently implement using fast Fourier transform (FFT) algorithms. This topic is given much greater coverage with figures explaining the aliasing

[3] `http://gnuradio.org/doc/doxygen`

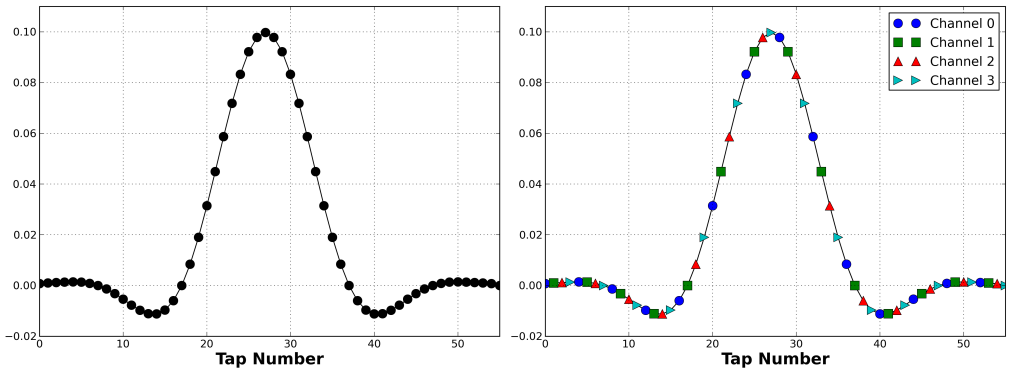

Fig. 1. Prototype filter partitioned into four filterbanks. Each filterbank filter is a phase-delayed version of the same filter.

in fred harris' book on the subject [2].

The synthesis filter behaves the same way in reverse. It uses an FFT to spin the data properly for the aliases to be separated and then filtered through the filterbanks. The output samples are then successively taken from the filterbanks.

Later, we will modify these structures to allow us to handle signals where the sampling rate is twice the channel bandwidth. The changes necessary to these structures is important and complicated. The work in [3], [4] explains the required changes.

IV. Prototype Filter Design in GNU Radio

In analysis and synthesis filterbanks, each channel is $1/M$th the total bandwidth of the received or transmitted, respectively, bandwidth. We either break the signal apart in the channelizer or put them together in the synthesizer. We do the same with the prototype filter. By partitioning the filter, we split the original filter into M pieces. So building the prototype means building a filter capable of filtering a single channel at the full bandwidth (i.e., the sample rate we receive or transmit on) of all M channels. This is important to understand that in a polyphase filterbank, the prototype filter is designed at the total bandwidth of all M channels.

GNU Radio has tools for building filters that we will use to construct the prototype filter, including a graphical program called *gr_filter_design*. In this application, we set the type of filter we want, which will be a windowed low pass FIR filter. We set the sample rate, the end of the pass band, and the start of the stop band along with the stop band attenuation (in dB). The pass band and stop band parameters are specified in units relative to the sample rate. Using the real sample rate of the full channel in samples per second, we can also specify the pass band and stop band parameters in Hertz. An alternative approach is to set these values using a normalized sample rate or 1 and make the other parameters relative to this.

TABLE I
Channelizer Parameters

Parameter	Value
Type	FIR
Style	Low Pass
Window	Hann Window
Sample Rate	20 MHz
Filter Gain	1
End of Pass Band	125 kHz
Start of Stop band	225 kHz
Stop Band Atten.	60 dB

The other parameter we set for this type of filter is the window type. In this section and in Figure 1, we use a Hann window because its parameters are well understood [1]. In latter parts of the paper, we found it easier to build perfect reconstruction filters using the Blackman-harris window [5].

Continuing with our broadcast FM example, we know that the full spectrum bandwidth is 20 MHz and each channel is 200 kHz. We make a lowpass filter based on this. However, we will not design this filter to have the full bandwidth of 200 kHz. The issue is that in the process of channelizing, we will alias bands together at the channel bandwidth, so filtering beyond the channel boundaries ends up aliasing in pieces of the neighboring channels. Instead, we will select parameters that will filter out the edges of our band with a transition period that ends roughly with the channel. The final parameters we select for this experiment are shown in Table I.

We made the start of the stop band slightly beyond the channel in order to give us a filter with a smaller number of taps and knowing that any aliasing with this will be negligible to the FM quality. Even still,

the resulting filter consists of 1091 taps, which seems quite long. However, since we separate this filter among a filterbank of 100 channels, each channel will actually consist of only 11 taps (i.e., $\lceil 1091/100 \rceil$). Computationally, these become very light-weight filters, especially if we can parallelize the computations. A bigger concern may be memory usage, depending on the platform on which the filterbank is implemented.

The synthesis filter design issues are the same as the analysis filter. We can easily take in 100 channels of frequency modulated audio and synthesize them together to broadcast 100 channels of FM.

V. Using a Polyphase Channelizer

In the next section, we will look more heavily into using the synthesis filterbanks. Here, though, we continue with the channelizer example from the prototype filter we built in Table I.

The GNU Radio flowgraph is shown in Figure 2. The signal is taken from a USRP[4], sampled at 20 Msps, and fed to the channelizer (*gr::filter::pfb_channelizer_ccf*) after some gain adjustment (*gr::analog::agc2_cc*). In this flowgraph, we take a single channel out for FM demodulation and simply discard the rest. We use the Busports[5] concept in the GNU Radio Companion to handle the large number of connections. The FM demodulation is done with a WBFM receive block (*wfm_rcv* Python block in *gr::analog*), and this is followed up by a resampler (*gr::filter::pfb_arb_resampler_ccf*) to get the sample rates correct for the audio sink (*gr::audio::sink*). A multiplier (*gr::blocks::multiply_const_cc*) is in there for volume control.

Setting the receiver to a center frequency of 97.9 MHz, we channelize the entire FM broadcast spectrum. The output of a slightly modified flowgraph from Figure 2 where we plot the output of two channels is shown in Figure 3. The input spectrum is shown along with channels 5 and 10. We only show the two channels here for space even though all 100 channels are available.

VI. Designing Reconstruction Filters

For the FM example, and other signals that have equal channel bandwidths and spacing, the standard

[4] http://ettus.com

[5] http://gnuradio.org/redmine/projects/gnuradio/wiki/Busports

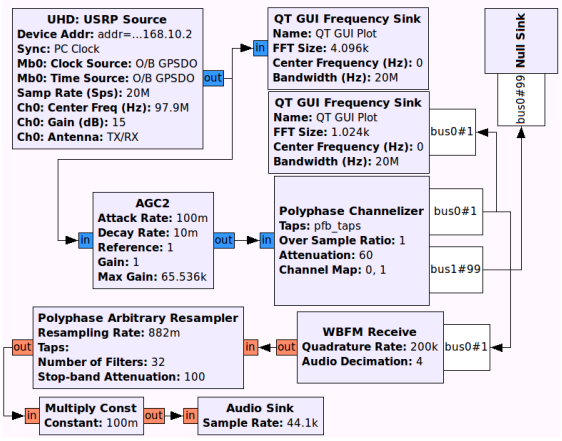

Fig. 2. GNU Radio flowgraph for channelizing the FM broadcast band and pulling out a single channel for FM demod.

analysis and synthesis filterbanks work well. But for cases becoming more relevant in developing ideas of DSA and cognitive radio, we may not have channels of equal spacing or bandwidth. For these cases, we need to modify the standard filterbank structures to allow for arbitrary bandwidth signals. Again, much of this work on the design of the filterbanks and the prototype filters is already published. We require few modifications on the filterbank designs, some modifications on how we partition the filters, and very specific design requirements of the prototype filter. Papers [3], [4] provide a good overview of the channelizer and synthesizer changes required to support the reconstruction and [6] provides details on the required changes to handling the filter partitioning in the updated synthesis filterbank.

First, the changes in the analysis and synthesis filterbank comes from the need to be able to completely *stitch* together two channels already split up by a channelizer filterbank without any loss. That means that, unlike before, we must design the filters such that the roll-off of the filter hits -6 dB just at the channel edge. Previously when channelizing, we designed the filter to make sure that it was within the bandwidth of the channel to prevent aliasing. Here, we have a requirement that when combining adjacent channels, the sum of the signals at the edges will recombine with the correct energy, which would introduce adjacent channel aliasing that we need to avoid.

To overcome the aliasing problem, we redesign the filterbanks to produce channels of the same bandwidth but now at twice the sample rate. With this, we move the adjacent channels far outside of our transition width of the filter, and so the signal remaining in the adjacent channels is specifically designed to hold

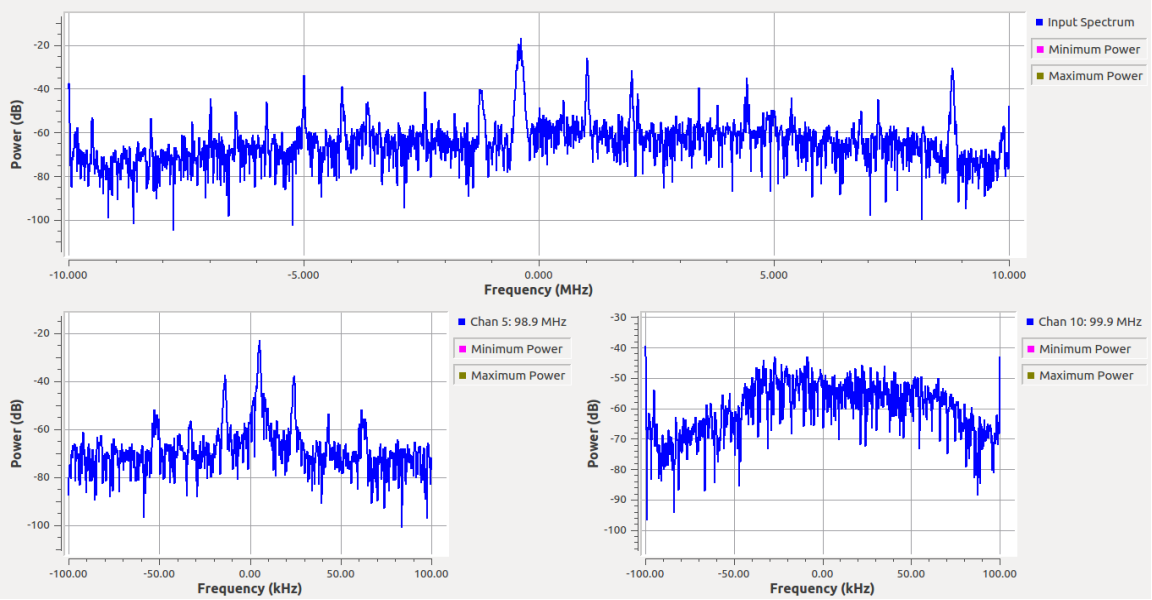

Fig. 3. Channels 5 (98.9 MHz) and 10 (99.9 MHz) displayed out of all 100 channels of the input FM band spectrum centered at 97.9 MHz.

the part of the current channel that spills over into the adjacent channel, but unaliased. From here, we can see that synthesizing these two channels, using the same prototype filter, will combine the signals such that the edges of both channels add together to create no loss between them.

Following from the channelizer now producing channels at twice the sample rate, we must also change the synthesis filterbanks to perform in a similar way. This structure takes in the channels at twice the sample rate and produces a signal that combines the channels now at twice the output sample rate. That is, synthesize N channels of $2f_s$ produces a signal at $2f_sN$.

To show how the reconstruction works, we will use an impulse as the input to the reconstruction filter, which is made up of a channelizer (*gr::filter::pfb_channelizer_ccf*) and synthesis filterbank (*gr::filter::pfb_synthesizer_ccf*) connected directly to each other as shown in Figure 4. The prototype filters for both filterbanks are the same except for a gain factor of $M/2$ (M is the number of channels) of the synthesis taps so that the output signal is at the same level as the input signals.

GNU Radio's *firdes* filter design program lets us fairly easily design the reconstruction filters we need. The *firdes* tool has a *low_pass_2* function to design a low pass filter with the gain, bandwidth, transition width, window function, and out-of-band attenuation as design parameters. To hit the required 6 dB roll-off at the channel's edge, we just need to set the bandwidth of our prototype filter equal to the bandwidth of the channel as the tool defines this as the end of the pass band. We then have the window, attenuation, and transition band as degrees of freedom to tweak our filter.

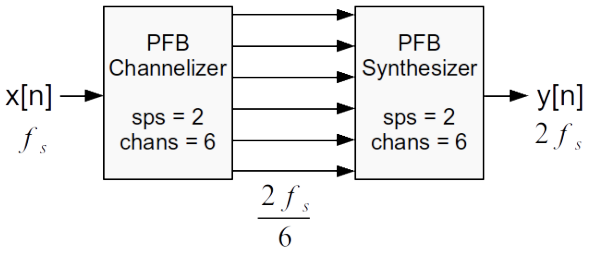

Fig. 4. Diagram of the reconstruction filter to split a signal into many pieces and reassemble them again.

In our analysis here, we will look at six channels with a sample rate of 2 kHz per channel. We lose no generality in the approach, but these few number of channels allows use to plot and visualize the process much easier than a large number of channels. Given this, our filter design parameters are shown in Table II.

Passing an impulse through this structure will result in an impulse out at the receiver at twice the sample rate. While this is an arbitrary example with a clean, known signal, the important consequence of this structure and the prototype filter design is that the resulting signal has been segmented and put back together with almost no change to the signal. The output, shown in Figure 5, shows that the signal is at

TABLE II

IMPULSE CHANNELIZER PARAMETERS

Parameter	Value
Sample Rate	12 kHz
Filter Gain	1* or 3**
End of Pass Band	1000 Hz
Start of Stop band	400 Hz
Stop Band Atten.	80 dB
Window	Blackman-harris [5]

* channelizer

** synthesizer

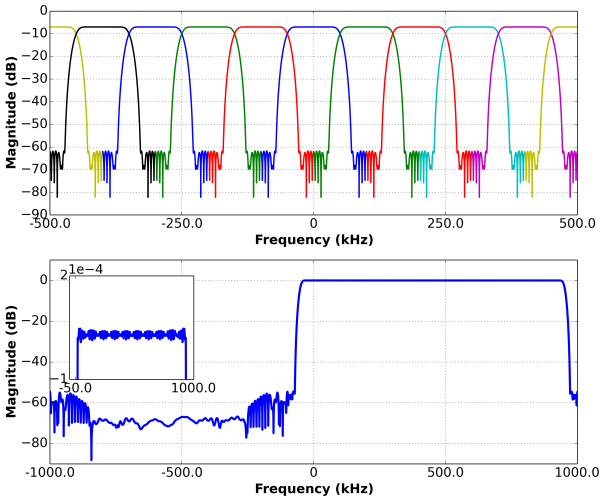

Fig. 5. Output of the reconstruction filter. Each channel filter is displayed in the top figure while the reconstructed signal is shown in the bottom figure.

twice the original sampling rate with very minor pass band distortion due to the filters.

Next, we show a more complex example that indicates better how this concept can be used. We stick with the impulse as the input signal to give a clean PSD of how the reconstruction produces perfect signals at the output. Instead of synthesizing all six channels back together, we use two synthesis filterbanks to combine the first four and last two channels. The design is shown in Figure 6 and the results in Figure 7. Not shown in these figures is the close-up of the reconstructed pass band; however, the two, four, and full six reconstructed channel examples all have the same maximum ripple of 5×10^{-4} dB. While the ripple is dependent on the original filter, the results do not change with the number of channels, making the technique scalable to large numbers of channels.

In this example, the more general use of the reconstruction filter becomes more obvious. Any signal can be broken up and reconstructed with a set of filters of

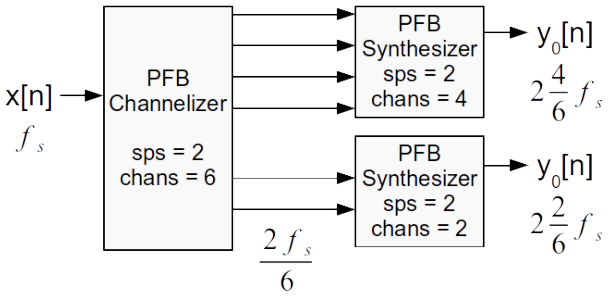

Fig. 6. Diagram of the reconstruction filter to split a signal into many pieces and reassemble into a set of four and two channels.

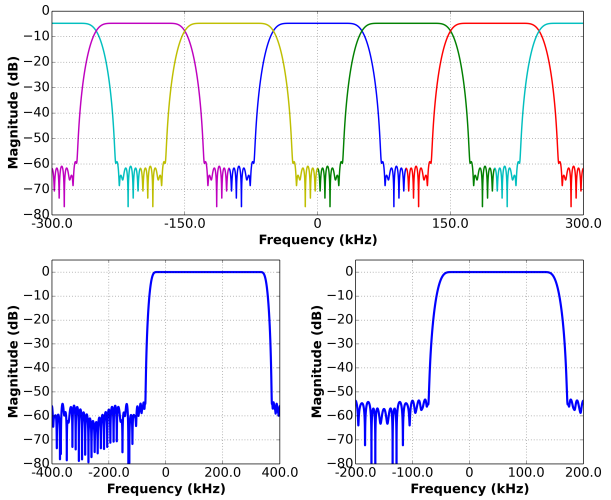

Fig. 7. Output of the reconstruction filter where an impulse is segmented into two signals of four and two channels.

this style, which means that we can now more easily channelize signals of different bandwidths and at different frequencies. One restriction comes into the design process, though. To work at twice the sample rate, both the channelizer and synthesizer must have an even number of filters or channels. This may mean using one more channel than required for some applications.

A. Channel Mappings

The GNU Radio channelizer and synthesis filterbanks have the ability to control the location of the output channels using a channel map, or permutation, through the *set_channel_map* function call. This mechanism provides a mapping of where each input channel will end up at the output of the filterbank. For the channelizer, the default mapping is that channel 0 is the 0 Hz channel, or the middle of the incoming frequency band. Channel numbers increase positively in the spectrum and wrap around to the negative half of the spectrum at $M/2$. For an even number of chan-

nels, the $M/2$ channel spans the positive and negative edges of the spectrum.

The channel map function takes a vector of numbers that is specified by an input channel value that will map to the output channel as the index of the array. This setup gives us flexibility in where and how to direct the channel positions, including being able to direct the same channel to multiple outputs.

The synthesis filterbank does the same: the index of the array is the output channel number and the value at that index is the input channel. Figure 8 demonstrates this. Remember that the output of the oversampling synthesis filterbank is at two times the sampling rate, which gives us twice the number of channels. By default, the channel mapping is from channel 0 to M for M input channels. All of the incoming channels from the channelizer are mapped directly to the corresponding output index value. That is why in the above graphs we saw the impulse covering the positive half of the spectrum; the other half of the channels produced by the synthesis filterbank had no input and therefore produce 0 on the output. In Figure 8, we are using the mapping sequence [10, 11, 0, 1, 2, 3]. The channels are mapped in order of their output from the channelizer, but we have used this mapping technique to shift the entire spectrum down by two channels.

B. Reconstructing FM

In this final example, we channelize part of the FM broadcast band into ten 100 kHz channels, select six of these to recombine, and demodulate the FM to audio. The GNU Radio flowgraph is shown in Figure 9 that uses a USRP B200[6] to pull in a 1 MHz signal centered at 101.3 MHz. There is a strong FM station at 101.5 MHz. We channelize the signal into ten channels using the parameters in Table III. The first six of these channels, covering the positive half of the spectrum, go into the synthesis filterbank. We terminate the other four channels for this experiment. The synthesizer parameters are shown in Table IV and are nearly the same as for the channelizer. Instead, though, this filter is designed at 600 ksps because we are only combining six 100 kHz channels together. The gain parameter is the half of the total number of channels in the channelizer, which normalizes the output. For convenience, we decimate the output by two using a very inexpensive half-band decimating filter. The rest of the flowgraph performs the FM demodulation and some signal processing to allow us to adjust the volume of the audio as well as resample it to match the required rate of the audio subsystem, just as we did above with the FM channelizer. And the synthesizer filterbank's channel map to set the resulting signal at DC is [10, 11, 0, 1, 2, 3].

[6] http://ettus.com

TABLE III
FM CHANNELIZER PARAMETERS

Parameter	Value
Sample Rate	1 MHz
Filter Gain	1
End of Pass Band	50 kHz
Start of Stop band	20 kHz
Stop Band Atten.	80 dB
Window	Blackman-harris [5]

TABLE IV
FM SYNTHESIZER PARAMETERS

Parameter	Value
Sample Rate	600 kHz
Filter Gain	5
End of Pass Band	50 kHz
Start of Stop band	20 kHz
Stop Band Atten.	80 dB
Window	Blackman-harris [5]

At different places in the flowgraph, we use frequency sinks to visualize the PSD of the signal. Figure 10 shows the output of these instrumentation tools, which shows the recovered signal with the standard FM in the middle as well as the HD radio sidebands. On the input spectrum, we added the channel boundaries of the ten channels to show how it was split up in the channelizer. In this case, we only require five of the channels to reconstruct the signal, but we use six to meet the requirement that the filterbanks must work with an even number of channels.

VII. CONCLUSION

From the details of this paper, we should understand the basics of how to design a polyphase filterbank prototype filter for analysis and synthesis filterbanks. This information gives us the basic knowledge of how to build filters with GNU Radio and use them in the GNU Radio polyphase filterbank blocks. We also discussed the added challenges of designing filterbank filters that allow perfect reconstruction, which we can used to break apart and then recombine channels as we need to in such a way that any signals on channel boundaries are reconstructed perfectly.

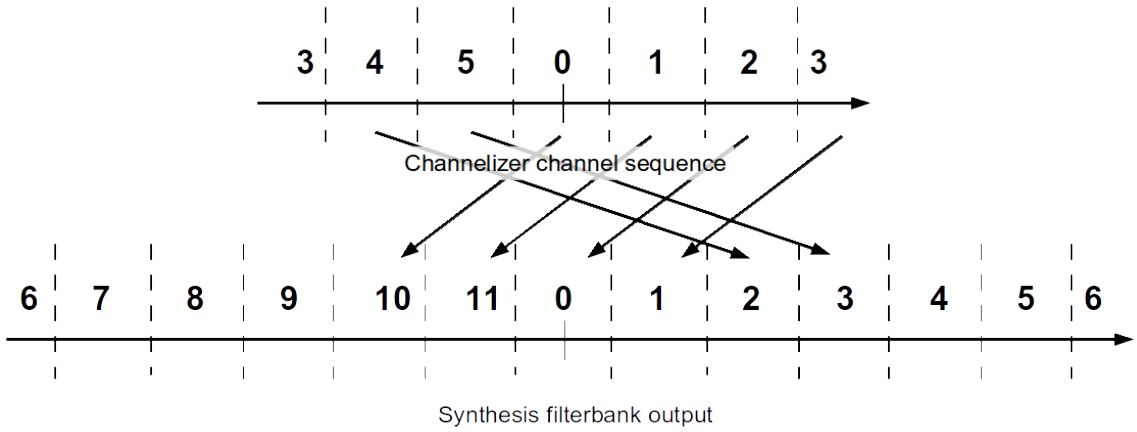

Fig. 8. Channel mapping in the synthesis filterbank using [10, 11, 0, 1, 2, 3] as a method of moving the spectrum down two channels.

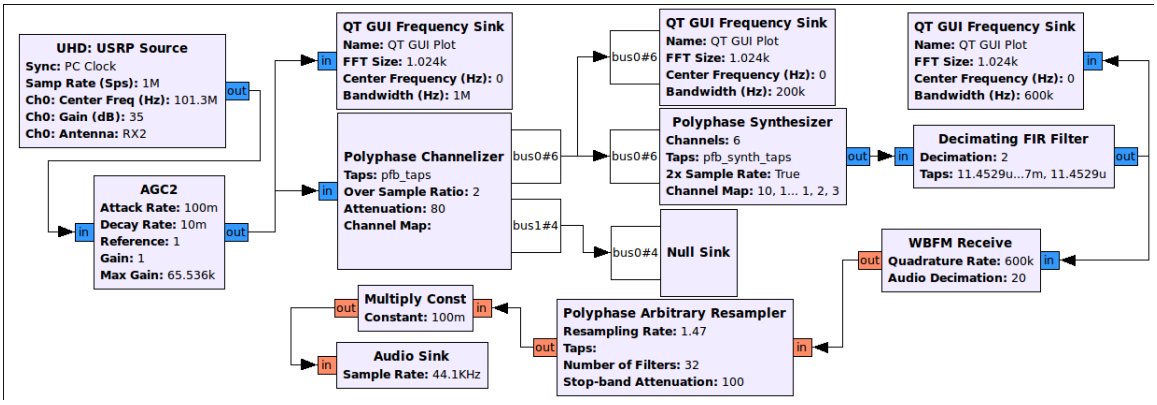

Fig. 9. GNU Radio flowgraph for channelizing and reconstructing a US broadcast FM radio station.

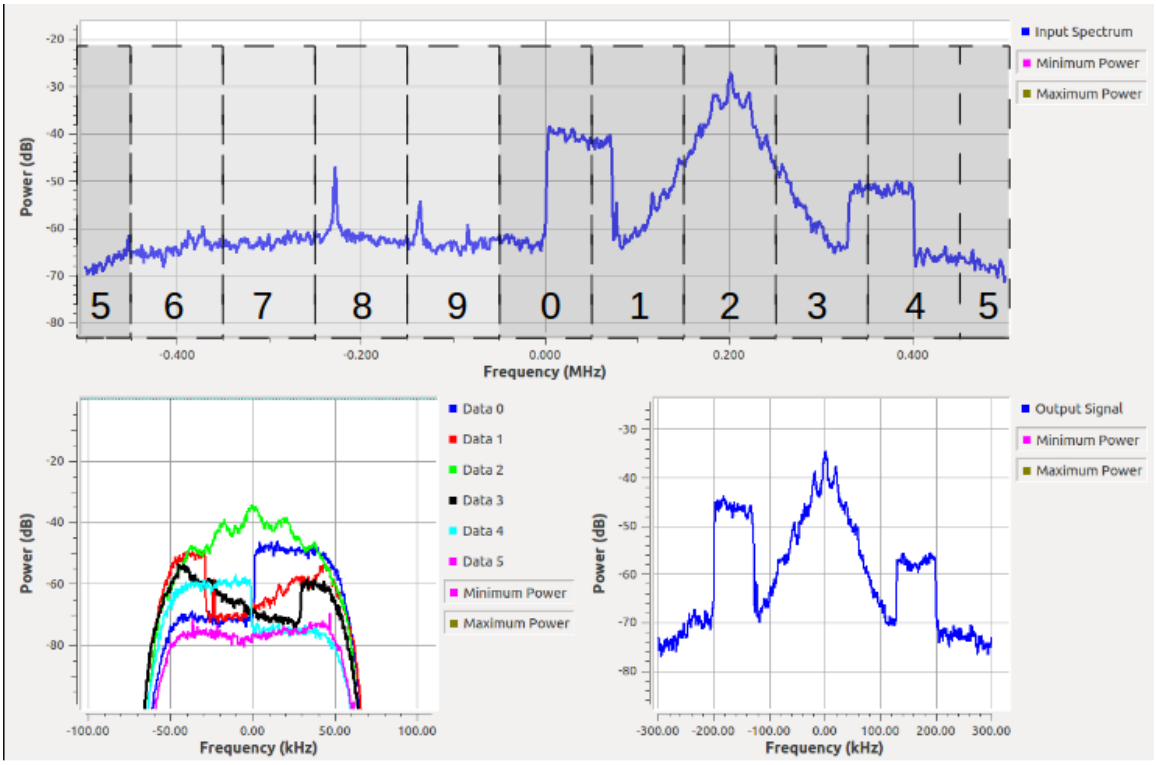

Fig. 10. Output of the reconstruction of the FM signal. This figure shows the input spectrum, the six highlighted channels sent to the synthesis filterbank, and the output of the filterbank and decimate-by-2 filter.

The details of how the filterbanks work is the subject of many other texts and papers referenced throughout. GNU Radio provides one of the few areas where the code, documentation, and examples exist to allow us to study and understand all of the aspects, features, and trade-offs of using these structures.

While this paper focused on the analysis and synthesis filterbanks, the same filter design principles are equally valid for any of the other polyphase filterbank tools we have in GNU Radio. Included among these are the arbitrary resampler and the clock timing synchronization blocks. In both of these filterbanks, we can set the quantization error by selecting the number of filters we use, which is set to 32 by default. In the terms used above, the number of channels is the number of filters in the filterbank. We then use the same concepts of designing the prototype filter by understanding this simple shift in nomenclature.

REFERENCES

[1] A. V. Oppenheim, R. W. Schafer, and J. R. Buck, *Discrete-time Signal Processing (2Nd Ed.)*. Upper Saddle River, NJ, USA: Prentice-Hall, Inc., 1999.

[2] fred harris, *Multirate Signal Processing For Communication Systems*. Upper Saddle River, NJ: Prentice Hall, 2004.

[3] E. Venosa, X. Chen, and fred harris, "Polyphase analysis filter bank down-converts unequal channel bandwidths with arbitrary center frequencies - design i," in *SDR'10-WinnComm*, 2010.

[4] X. Chen, E. Venosa, and fred harris, "Polyphase analysis filter bank up-converts unequal channel bandwidths with arbitrary center frequencies - design ii," in *SDR'10-WinnComm*, 2010.

[5] f. j. harris, "On the use of windows for harmonic analysis with the discrete fourier transforms," in *Proc. IEEE*, vol. 66, Jan. 1978.

[6] f. j. harris, C. Dick, X. Chen, and E. Venosa, "Wideband 160-channel polyphase filter bank cable tv channeliser," in *IET Signal Processing*, 2010.